MICRO CONSULT

08/2012

Die drei Gesichter der Sicherheit von Software-Systemen

Teil 3 –Die Zukunftssicherheit

In einer 3-teiligen Beitragsserie berichten wir über die Ergebnisse und Erkenntnisse des Embedded Software Engineering Kongress 2011. In den ersten beiden Teilen ging es um die Betriebs- und die Angriffssicherheit. Im heutigen Beitrag geht es um den etwas schwieriger zu definierenden Begriff der Zukunftssicherheit, der sehr viele Aspekte beinhaltet. Die Zukunftssicherheit beruht nicht nur darauf, dass die Anpassung an künftige Anforderungen der Betriebs- und Angriffssicherheit, der Funktionalität, des Designs und der Standards auch in Zukunft wettbewerbsfähig erfolgen kann.

Der Gedanke liegt nahe, dass es hellseherischer Fähigkeiten bedarf, um heute schon sagen zu können, welchen Gefahren und Herausforderungen die aktuelle Embedded-Software in der Zukunft ausgesetzt sein wird. Doch zum Glück gibt es auch ganz diesseitige Methoden und Hilfsmittel, die uns, konsequent angewendet, diesem Ziel näher bringen können.

Ein prominentes Beispiel zeigt, wie wichtig die Zukunftssicherheit sein kann: 1977 starteten von Cape Canaveral 2 Trägerraketen, beladen mit den Voyager-Raumsonden. Sie hatten die Aufgabe, verschiedene Planeten unseres Sonnensystems zu erkunden. Die geschätzte Projektlaufzeit betrug rund 5 Jahre. Alles musste über diese Lebensdauer störungsfrei funktionieren. Das Voyager-Programm funktioniert viel besser als erwartet. Bis heute funken die Sonden ihre Signale, rund 30 Jahre länger als geplant. Die verantwortlichen Techniker haben ihre Sache offensichtlich gut gemacht. Das System hatte allerdings einen großen Vorteil: Es blieb über seine Lebensdauer unverändert.

Wie es allerdings ausgehen kann, wenn Software häufigen Änderungen unterworfen ist, zeigte sich beim Smartphone-Betriebssystem Symbian. Es hatte einige Jahre sehr gute Dienste geleistet, doch nach und nach begann es, instabil zu werden. Auch bei kleinen Änderungen wurde immer schwieriger, die Software wieder zu stabilisieren. Heute verschlänge die Weiterentwicklung so große Ressourcen, dass mit neuen Versionen nicht mehr zu rechnen ist. Diese veränderungsbedingte Fragilität nennt man Softwareerosion.

Softwareerosion gehört damit zu den größten Bedrohungen für die schnellen Innovationzyklen in vielen Branchen, wie Thomas Eisenbarth von der Firma Axivion und Prof. Dr. Koschke betonen. Beide befassen sich seit mehreren Jahren mit gezielten Maßnahmen gegen Softwareerosion. Sie sehen in einem besseren Problembewusstsein und in automatisierten Erosionstests in Verbindung mit konsequenten Gegenmaßnahmen wichtige Voraussetzungen. Rudolf Frommknecht, Squoring, ergänzt, welche Bedeutung Ausbildung, Prozessgestaltung und Toolauswahl haben.

Heute sind die Herausforderungen an Softwareentwickler für Embedded-Systeme, um Zukunftssicherheit bei ihren Produkten hinsichtlich Softwareerosion und auch allgemein zu gewährleisten, bekannter als noch vor wenigen Jahren.

Generell gilt der Grundsatz: Keep it simple. Die Vermeidung undurchsichtiger Konstrukte und die Wahl eines Softwarecodex, der weitgehend selbsterklärende und überschaubare Softwareelemente fordert, ist ein guter Einstieg in diese Zukunftssicherung.

Qualitätsbeurteilung

Jede Software, die sozusagen aus der Hand eines Programmierers geflossen ist, kann gewissermaßen in ihrem "so sein, wie sie ist" beurteilt werden. Dieses "so sein" wird landläufig mit dem Begriff der Qualität bezeichnet. Qualitätsmodelle helfen dabei, spezifische Qualitätskriterien von Produkten zu beurteilen. Klaus Wissing von Squoring erklärte auf dem ESE Kongress, dass bei der Softwareentwicklung immer mehr Personen mit ganz unterschiedlichen Rollen und Blickwinkeln beteiligt sind. Die Aufgaben innerhalb der Prozesse sind einerseits hochgradig vernetzt, müssen andererseits aber auch gegeneinander abgegrenzt werden. Die Vielfalt der sich daraus ergebenden Projektanforderungen zu beherrschen, ist die große Herausforderung. Jeder Projektbeteiligte muss dazu einen realistischen Einblick in den Projektfortschritt und die Softwarequalität erhalten, der seinem Informationsbedürfnis und seinem Kenntnishorizont entspricht. Idealerweise wird diese Information per Knopfdruck auf Basis definierter Qualitätsmodelle automatisch ermittelt und visualisiert. Die große Kunst besteht nun darin, geeignete Qualitätsmodelle zu erstellen und diese auch an veränderte Bedingungen anzupassen.

Bekannte und verbreitete Standards, Normen und Metriken dienen als Grundlage für solche Qualitäts- und Bewertungsmodelle. Der konkrete Fall ist eingehend zu betrachten und angepasste Tools und Maßnahmen sind einzusetzen. In Bezug auf die Beurteilung von Zukunftssicherheit würde das bedeuten, dass eine Software und der damit verbundene Prozess anhand spezifischer Kriterien, z.B. der Updatefähigkeit, der Wirksamkeit des Change-Managements oder der Stabilität, nach Änderungen geprüft werden.

Software-Lebenszyklen

Die Zukunftssicherheit einer Software endet aber nicht mit dem erfolgreichen Abschließen des "Lebensabschnitts" Implementierung oder Test. Die Zukunft beginnt ja quasi erst in der Zukunft, das Urteil lautet demnach: Lebenslänglich! Zu dem Schluss kamen Dr. Michael Sturm und Carolin Eckl von der Berner und Mattner Systemtechnik GmbH bei ihrem Vortrag über "Architecture Lifecycle-Optimierung mit Embedded Design Pattern".

Der Fokus darauf ist einem Wandel unterworfen, der sich gerade in der letzten Zeit immer stärker bemerkbar macht. Wo früher aufgrund limitierter Leistungsfähigkeit und Kapazität der Hardware jeder Entwickler einer Embedded-Software in der Gestaltung seines Produkts massiv eingeschränkt war, nähert sich die Embedded-Programmierung der für PCs an. Wartbarkeit und Flexibilität sind heute wichtiger denn je. Und die Beherrschung einer zunehmenden Variantenvielfalt, die Forderung nach gut und ansprechend gestalteten Nutzeroberflächen sowie die umfassende Vernetzung von Systemen ergänzen den Katalog der aktuellen Anforderungen. Die Entwicklung von Embedded-Software ist daher heute viel komplexer als noch vor nicht allzu langer Zeit. Aus dem Grund sind umfangreichere Maßnahmen erforderlich, die beispielsweise die Verwendung neuester, objektorientierter Programmiersprachen oder Modellierungsmethoden erfordern. Darüber hinaus sind auch neue Rollen und Aufgaben, wie z.B. die des Softwarearchitekten sowie des Konfigurations-, Versions- und Variantenmanagements, notwendig.

Offene Codes

Open Source Software mit ihren vielen Entwickler, die daran arbeiten, es weiterentwickeln und voranbringen, gewährleisten die Sicherheit, Aktualität und hohe Qualität des Systems, auch zukünftig. Projekte und Produkte aus dem Open Source Umfeld weisen eine Besonderheit auf: Jeder, der dazu in der Lage ist, kann ein vorhandenes System verändern und verbessern. Die dafür erforderlichen Dateien stehen öffentlich zur Verfügung. Sie unterliegen der sogenannten GNU General Public License (GPL) und dem "Copyleft" der GPL. Alle Änderungen, die vorgenommen werden, müssen anschließend ebenfalls veröffentlicht werden, um ggf. erneut weiterentwickelt oder verbessert werden zu können. Allerdings kann dies dem Interesse, sein geistiges Eigentum zu schützen, entgegenstehen.

Dr. Carsten Emde, Open Source Automation Development Lab (OSADL), referierte auf dem ESE Kongress über das Thema, ob man nun seinen Code tatsächlich offenlegen muss oder nicht. Er räumt in diesem Zusammenhang dem Begriff der Ableitung eine zentrale Bedeutung ein. Die Frage, ob eine Software abgeleitet wurde, ist entscheidend dafür, wem die Software letztlich tatsächlich gehört. Wer vom Geben-und-Nehmen-Prinzip der Open Source Gemeinde profitieren will ohne Schlüssel-Kowhow preisgeben zu müssen, sollte sich an dieser Stelle rechtzeitig über die lizenzrechtlichen Implikationen informieren.

Innovation

Die sprichwörtliche Aussage, dass man den Wald vor lauter Bäumen nicht sieht, erfüllt sich leider auch hinsichtlich der Entwicklung von Software für Embedded-Systeme. Anforderungen aus Lasten- und Pflichtenheften, gesetzliche Vorgaben, Normen und Richtlinien geben die Marschrichtung vor. Das klingt wenig verlockend, weil umständlich und schwierig, ist aber zumindest ein sicherer Weg. Die Chancen, aber auch Risiken neuer Wege zeigen sich meist aber erst dann, wenn diese begangen werden. Von Woody Allen stammt hierzu das treffende Zitat: "Wenn Sie nicht von Zeit zu Zeit auf die Nase fallen, ist das ein Zeichen, dass Sie nichts wirklich Innovatives tun".

Das mag wie eine Binsenweisheit erscheinen, doch die Frage ist, wo die Kreativität ansetzen soll. Bekannt sind heute Kreativitätstechniken, wie z.B. die Osborn-Checkliste, die einem Entwickler die Möglichkeit gibt, eine definierte Aufgabe nicht nur einfach kreativ zu lösen, sondern schon die Definition der Aufgabe kreativ anzugehen. Jean-Philippe Hagmann von Denkmotor erklärte dazu, dass Kreativität gemäß der Wortherkunft ja nichts anderes als "Denken" bedeutet; Denken allerdings, das zu neuen Lösungen führt. Eine zentrale Aussage ist, dass Kreativität für den Fortschritt und somit für die Zukunft von großer Bedeutung ist. Hat ein Entwickler die Aufgabe, eine Embedded-Software zu entwerfen, so ist er gut beraten, das momentan nur Wägbare dennoch so gut es geht zu berücksichtigen. Dazu gehört auch Mut, denn das heute Unbekannte kann auch Risiken bergen. Die Osborn-Checkliste liefert gerade in der Ideenphase hilfreiche Impulse für alternative Verwendungen, Anpassungen oder Abwandlungen.

Zukunftssicherheit durch Kompetenz

Die Zukunftssicherheit von Software spiegelt sich in besonderem Maße in der Zukunftsfähigkeit ihrer Produktmanager, Architekten und Entwickler wider. Wer heute Software entwickelt, sollte seinen Blick in die Zukunft richten. Dabei lohnt sich der Blick über den „Embedded-Tellerrand§ in die große, weite Welt der IT. Welche Methoden und Tools werden dort heute eingesetzt, und welche Bedeutung könnten sie für die Embedded-Welt von Morgen haben? Hellseherei ist dabei gar nicht nötig; es genügt meist, vor die Tür zu treten und die Augen zu öffnen. In jedem Falle ist es empfehlenswert, sich nach Technologien und Darstellungsmöglichkeiten umzusehen, die von Programmiersprachen unabhängig sind. Solche Ansätze erleichtern es, Architektur, Ideen und Prinzipien einer Software ohne aufwändige Sourcecode-Sezieranstrengungen in die Zukunft mitzunehmen.

Zusammenfassung

Dieser Beitrag kann naturgemäß nur einen groben Überblick über das so große Thema der Zukunftssicherheit geben. So wenig es möglich ist, die Betriebssicherheit einer Embedded-Software oder ihren Schutz vor Angriffen von heute an für alle Zeiten und alle erdenklichen Fälle mit 100%iger Sicherheit zu gewährleisten, so wenig ist das für die Zukunftssicherheit möglich. Allerdings erhöht die bewusste und kritische Auseinandersetzung mit Themen wie Softwareerosion, Qualitätsmodellen, Open Source, Innovation und Wissensmanagement die Chance für eine zukunftsfähige Embedded-Software.

Autor:
Peter Siwon ist Business Development Manager beim Münchner Unternehmen MicroConsult, Trainingsspezialist für Embedded Software Engineering.

Alexander Sedlak ist als freier Autor tätig.

MicroConsult GmbH:
Training, Coaching und Consulting für Software- und Hardwareentwicklung sowie Führungskräfte in der Industrie.

www.microconsult.de